**Université Ibn Khaldoun – Tiaret**
**Faculté des Mathématiques et Informatique**
**Département d'Informatique**

**Matière : ASDD2**
**Niveau : 1 LMD (S1, S2, S3 et S4)**
**Année Universitaire : 2021-2022**
**Durée : 01$^{H}$:00$^{mn}$**

# **Examen Final** (Semestre 2)

**Exercice 1** (*6 pts*). Donnez le résultat d'exécution de la fonction récursive « DS(1111) » et devine ce qu'elle fait.

```
int DS(int n1) {
      if (n1 == 0)
            return 0;
      return ((n1 % 10) + DS(n1 / 10));

}
```

**Solution :**

- Résultat d'affichage de DS(1111) est : « **4** ».
- La fonction retourne **la somme des chiffres qui composent un nombre donné**.

**Exercice 2** (*7 pts*).

Une carte grise contient les informations sur un véhicule : Nom et prénom du propriétaire, adresse, N° matricule, marque, type, puissance, année, Couleur.

Le matricule (en Algérie) est un code contenant quatre informations : un numéro de série, le type, l'année, et la wilaya.

1) Donner la structure carte grise.
2) Créer un tableau de 100 véhicules pour remplir les champs déclarés.
3) Ecrire un algorithme qui compte le nombre de véhicules de la wilaya 31 et affiche ce résultat.

**Solution**

```
typedef struct NumMatricule NumMatricule;
typedef struct NumMatricule {
      int numSerie, type, annee, wilaya;
};
typedef struct CarteGrise CarteGrise;
typedef struct CarteGrise {
      char nom[20], prenom[20], adresse[20];
      NumMatricule matricule;
      char marque[20], type[20];
      int puissance, annee;
      char Couleur[20];
};

int main() {
//    créer un tableau de 100 véhicules;
      CarteGrise tabVehicule[100];
      int i;
      int nombreVehicule = 0;
//    Lecture du tableau
      for (i = 0; i < 100; i++) {
            scanf("%s%s%s", &tabVehicule[i].nom, &tabVehicule[i].prenom,
                        &tabVehicule[i].adresse);
```

```
            scanf("%d%d%d%d", &tabVehicule[i].matricule.numSerie,
                        &tabVehicule[i].matricule.type,
&tabVehicule[i].matricule.annee,
                        &tabVehicule[i].matricule.wilaya);
            scanf("%s%s", &tabVehicule[i].marque, &tabVehicule[i].type);
            scanf("%d%d", &tabVehicule[i].puissance,
&tabVehicule[i].annee);
      }
//  L'affichage du tableau
      for (i = 0; i < 100; i++) {
            printf("%s %s %s \n", tabVehicule[i].nom,
tabVehicule[i].prenom,
                        tabVehicule[i].adresse);
            printf("%d %d %d %d \n", tabVehicule[i].matricule.numSerie,
                        tabVehicule[i].matricule.type,
tabVehicule[i].matricule.annee,
                        tabVehicule[i].matricule.wilaya);
            printf("%s %s  \n", tabVehicule[i].marque,
tabVehicule[i].type);
            printf("%d %d\n", tabVehicule[i].puissance,
tabVehicule[i].annee);
      }
//    Nombre de véhicule de la wilaya 31
      for (i = 0; i < 100; i++)
            if (tabVehicule[i].matricule.wilaya == 31)
                  nombreVehicule++;
      printf("Le nombre de véhicules de la wilaya 31 est %d\n",
nombreVehicule);

      return 0;
}
```

**Exercice 3** (*7 pts*).

Nous voulons enregistrer les valeurs de **N** nombres complexes (**N > 0)** à l'aide d'une LLC.

En utilisant des procédures/fonctions et les enregistrements, écrire l'algorithme/Prg en C qui permet de :

1) Créer une liste des nombres complexes.
2) Afficher le résultat d'une fonction « **SumL_Z**() » (qui calcule la somme des nombres de la liste précédemment créée).
3) Afficher le résultat d'une fonction « **MoyL_Z**() » (qui calcule la moyenne des nombres de la liste précédemment créée).

**Rq :** Si $Z1 = X1 + Y1\,\underline{i}$ et $Z2 = X2 + Y2\,\underline{i}$ alors $Z1+Z2 = (X1+X2) + (Y1+Y2)\,\underline{i}$.

**Solution**

```
Algorithme EXO2 ;
Type Tcomplexe = enregistrement
       PR : reel ;
       PI : reel ;
       FinEnreg ;

      TLcomplexe = enregistrement
       PR : reel ;
       PI : reel ;
       Suivant: ^TLcomplexe ;
       FinEnreg ;

Var Tete : ^ TLcomplexe ; N : entier ;
```

```
Procedure CreerL(var T : ^ TLcomplexe ; M : entier) ;
Var p : ^ TLcomplexe ; i : entier ;
Debut
        Pour i ← 1 à M faire
        Nouveau(P) ;
        Lire (p^.PR, p^.PI) ;
        p^.suivant ← T ; T ← p ; p ← Nil ;
        FFpour ;
Fin ;

Fonction SumL_Z(T : ^ TLcomplexe ) : Tcomplexe ;
Var p : ^ TLcomplexe ; S_R, S_I : reel ;
        resultat: Tcomplexe ;
Debut
        S_R ← 0; S_I ← 0; p ← T;
        Tanque(T <> Nil) faire
                S_R ←S_R + p^.PR ;
                S_I ←S_I + p^.PI ;
                p ← p^.suivant;
        FTQ;
        resultat.PR ← S_R; resultat.PI ← S_I;
retourner resultat;

Fin;

Fonction MoyL_Z(T : ^ TLcomplexe ) : Tcomplexe ;
Var p : ^ TLcomplexe ; S_R, S_I : reel ;
        resultat: Tcomplexe ; cmpt: entire;
Debut
        S_R ← 0; S_I ← 0; p ← T; cmpt ←0;
        Tanque(T <> Nil) faire
                S_R ←S_R + p^.PR ;
                S_I ←S_I + p^.PI ;
                Cmpt ← cmpt +1 ;
                p ← p^.suivant;
        FTQ;
        resultat.PR ← S_R/ cmpt ;
        resultat.PI ← S_I/cmpt ;
retourner resultat;

Fin;

Debut
        Repeter Lire(N) ; Jusqu'a (N>0) ;
        Tete ← Nil ;
        CreerL(Tete, N) ;
    Ecrire('la somme de la liste : ',
SumL_Z(Tete, N).PR,'+', SumL_Z(Tete, N).PI,'i' );

    Ecrire('la Moyenne de la liste : ',
MoyL_Z(Tete, N).PR,'+', MoyL_Z(Tete, N).PI,'i' );

Fin.
```